Path Difference Learning for Guitar Fingering Problem

Aleksander Radisavljevic and Peter Driessen

Department of Electrical and Computer Engineering, University of Victoria, British Columbia
peter@ece.uvic.ca
radis@pacificcoast.net

Abstract

In this paper we address the problem of mapping guitar music score into one of possible alternative fingering sequences on the fretboard grid. We use dynamic programming (DP) to model the decision process of a guitarist choosing the optimal fingering sequence. To estimate the DP cost functions based on examples of guitar fingering transcriptions (tablatures) we developed an original method named "path difference learning" employing a gradient descent search on the coefficients of the cost function. Features of the fingering alternatives that capture the essence of the mechanical difficulty and musical quality are used to reject impractical fingerings and thus reduce the DP search complexity. Our experiments for several classical guitar pieces showed consistent convergence of the path difference learning. The adaptation resulted in a significant decrease in error count compared to manually selecting the cost function weights.

1 Introduction

String instruments with fixed fret positions such as guitar offer multiple alternative positions, i.e. string-fret combinations where a given note can be played. Each position can furthermore be played by any of the four left-hand fingers and considering that multiple notes are played at the same time, it becomes apparent that there exists a large number of possible fingering alternatives for a single set of notes. As the guitarist executes a song she selects one of the large number of possible paths that minimizes the mechanical difficulty of the resulting sequence including the transition difficulty from one stage (set of notes) to another. In addition to the ease of play the player evaluates and selects the fingering sequence based on musical criteria such as matching the acoustical quality of all vibrating strings (Sayegh 1989). Overall, this transcription process presents a significant analytical challenge for most novice to intermediate players. To address this problem the guitar music notation evolved to include occasional fingering information such as fret number or finger label. Another more effective solution came with the emergence of guitar tablature (Phillips, Chappel and Chappel 1998) where the six line notation represents guitar strings and the superimposed numbers indicate the frets of the played notes.

The idea of applying computer-based expert systems toward the guitar fingering problem is found in (Sayegh 1989) which reports a number of rules used by expert guitarist in decision making and proposes the optimum path paradigm as a suitable method for applying these criteria and calculating the optimal fingering solution. The optimum path paradigm is equivalent to dynamic programming (DP) in its special case for deterministic systems (Bertsekas 1987). The main challenge in the use of DP for guitar fingering is the lack of an explicit cost function. For example, weighing the cost function to favor large changes in fret positions vs. movement of fingers to different strings is a difficult and highly subjective decision to make. Furthermore, it is known that guidelines for fingering guitar music differ for various guitar styles, such as Baroque, Folk, Blues, (Gilardino 1975), (Phillips, Chappel and Chappel 1998) as well as individual playing styles. Each of these styles would therefore require a different cost function. To avoid the time consuming process of interviewing expert guitarists and manually adjusting cost function weights we developed a method of learning cost function weights from published tablatures (Harris, 1999), (OLGA, 2004). Since this learning procedure seeks to minimize the difference between the desired and the optimal path we select the term Path Difference (PD) learning .

2 Optimal Path Solution

A sequence of notes defines a song which is the input information for the guitar transcription problem. We define discrete times $k=[0,1,2,\ldots,N]$ as the locations in the sequence where one or more notes change, due to an onset or release of a note . These times define the $N+1$ stages of the optimal path search graph. Each stage k therefore presents a set of notes that needs to be executed on the instrument's string-fret grid. As notes can generally be played in a number of different locations using different fingers each set of notes defines a set of possible fingering alternatives $S_k = \{G_{k,0}, G_{k,1}, \ldots, G_{k,M^{k-1}}\}$ which will also be referred to as states. These states $G_{k,l}$ can be represented as a matrix consisting of one row (string,fret,finger) for each note (Figure 1).

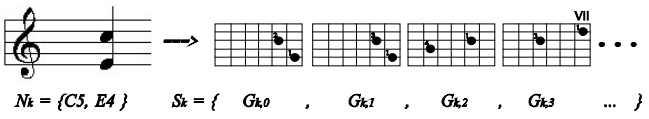

Figure 1. An example of generating polyphonic fingering alternatives for a single note set {C5,E4}. Symbols correspond to fretboard graphics directly above. The numbers located above each fret-string position identify the finger used.

Given a list of choices at each stage *k* the transcription process selects the fingering alternative from set S_k as described in Section 1. To measure these criteria we define a transition cost function $C_t(i,j)$ and static cost function $C_s(i)$. The former defines the cost of transition between two consecutive stages, i.e. from state $i \in S_k$ to state $j \in S_{k+1}$. Static cost function takes a single state *i* as the argument. The variables *i* and *j* are used as a more compact notation for states taken from sets S_k. A fingering sequence incurs a total path cost equal to the weighted sum of static and transition cost terms for the states occurring on the path (see figure 2). The static cost term is a novel feature we introduce primarily to model the varying static difficulty of different fingering alternatives when multiple notes are played simultaneously.

With cost functions specified we are now able to search for the optimal path as illustrated in the state transition graph (figure 2). Dynamic programming algorithm performs this search efficiently in a stage-by-stage manner based on Bellman's principle (Bertsekas 1987). Exhaustive search, by comparison, would require evaluation of all possible paths through the transition graph. The dynamic programming algorithm proceeds backward in a recursive fashion according to the following equation,

$$J_N(i) = C_s(i), \quad i \in S_N \tag{1}$$

$$J_k(i) = \min_{j \in S_{k+1}} \{C_t(i,j) + J_{k+1}(j)\} + C_s(i), \quad i \in S_k$$

$$for \quad k = N-1, N-2, ..., 0 \tag{2}$$

where $J_k(i)$ is the minimum cost from state $i \in S_k$ to the terminal node. Retracing this path for $J_0(0)$, i.e. from initial state yields the globally optimal path.

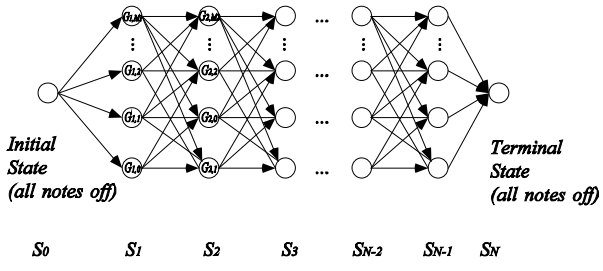

Figure 2. The state transition graph. The number of states will in general be different at each *k*.

3 Using Feature Representation for Fingering Alternatives

From the definition of states G we observe that there is a large number of possible polyphonic states achievable within physical constraints of the left hand. Each combination of up to 6 positions on a string-fret grid of 6x17 and each combination of associated fingers results in a unique state. The approach of assigning a static cost to each state and a transition cost to all possible state transitions results in prohibitive memory requirements, since calculating transition cost directly in the state domain would require a lookup table with an entry for each combination of states (*i,j*). Instead we seek to extract *features* which contain the information to discriminate between desired and undesired states. Examples of transition features are "number of frets traversed by a specific finger", "finger changes from used to unused", etc. Example of static features are "number of frets between consecutive fingers", "average fret location", "number of empty strings ", etc. The cost functions can then be expressed in terms features extracted from states as following,

$$C_t(i,j) = F_t(i,j) \cdot w_t \tag{3}$$

$$C_s(i) = F_s(i) \cdot w_s \tag{4}$$

where feature extraction functions $F_t(i,j)$ and $F_s(i)$ result in vectors of transition and static features respectively and weights determine the "relative-importance" of individual features. In the linear case expressed in (3),(4), the cost is simply an inner product between feature vectors F_t , F_s and the weight vectors w_t, w_s respectively. Since features nicely describe important physical aspects of the fingering alternatives they can be effectively used to reject impossible states from the combinatorial expansion described in figure 1. This can further greatly reduce the DP search space.

4 Path Difference Learning

PD learning requires a training set which consists of an input sequence of notes and the corresponding fingering sequence selected by an expert guitarist. We shall refer to the later as the *desired path* within the dynamic programming transition graph. The desired path, which can readily be obtained from guitar tablatures, represents a playing style we seek to model by adjusting the weights of the cost functions. The resulting weights determine the relative importance of individual cost measures for this playing style. The goal of this learning phase is to use the trained system to generate transcriptions in the same fingering style for other guitar compositions.

The main idea behind PD learning is to adjust the cost function weights until the desired path becomes optimal within the dynamic programming search (1),(2). To achieve this search in the cost function weight space it would be convenient to apply some stochastic search method such as simulated annealing (Kirkpatrick, Gelatt and Vecchi 1983)

or genetic search (Goldberg 1990), this however results in prohibitive computational complexity as the dynamic programming search is evaluated in each trial. In contrast, PD focuses only on sections where the optimal and the desired paths are different. In this approach only the two competing paths are processed placing the focus on states "where the differences matter" (figure 3). Consequently the computational burden is much lower . The indices within the states in figure 3 are used as the state identifiers and may be used as arguments *i,j* for calculating costs C_s and C_t,

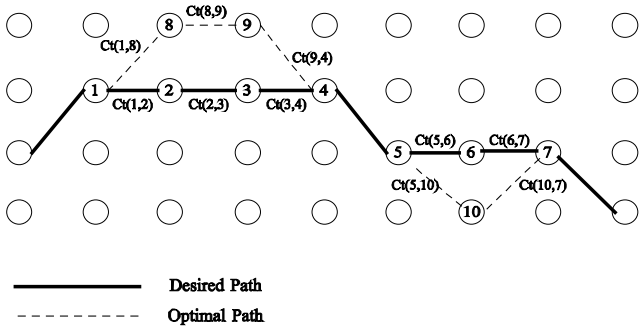

Figure 3. State transition graph with two path difference segments denoted as pairs {desired path, actual path} $\{P_d(0),P_a(0)\}=\{(1,2,3,4),(1,8,9,4)\}$ and $\{P_d(1),P_a(1)\} = \{(5,6,7),(5,10,7)\}$.

In the above example PD learning updates the weights of cost functions C_t and C_s such that the cumulative cost of the desired path becomes lower than the cumulative cost of the actual (previously optimal) path.

PD learning is a form of gradient descent update on cost function weights such that the total path cost of the desired path is decreased while the total path cost of the current optimal path is increased until the desired path becomes optimal. To avoid complex notation for the general case we illustrate this process using the example from figure 3 containing the two path difference.

Cumulative path costs before weight updates is:

$$\left.\begin{aligned} P_d(0) &= C_t(1,2,w_t)+C_s(2,w_s)+C_t(2,3,w_t)+C_s(3,w_s)+C_t(3,4,w_t)\\ P_a(0) &= C_t(1,8,w_t)+C_s(8,w_s)+C_t(8,9,w_t)+C_s(9,w_s)+C_t(9,4,w_t)\end{aligned}\right\} \quad P_d(0) > P_a(0) \tag{5}$$

$$\left.\begin{aligned} P_d(1) &= C_t(5,6,w_t)+C_s(6,w_s)+C_t(6,7,w_t)\\ P_a(1) &= C_t(5,10,w_t)+C_s(10,w_s)+C_t(10,7,w_t)\end{aligned}\right\} \quad P_d(1) > P_a(1) \tag{6}$$

The weight update equations are derived by minimizing an error measure E designed to increase as the ratio of actual and desired path costs increase. This error measure is defined as,

$$E(w_t,w_s)=\sum_j \begin{cases}\left(1-\dfrac{P_a(j)}{P_d(j)}\right) & \text{if} \quad P_d(j)>P_a(j)\\ 0 & \text{otherwise}\end{cases} \tag{7}$$

The update equations for transition and static weights then become:

$$w_t^{k+1}=w_t^k-\alpha\cdot\sum_{j=0}^{1}\left(P_d(j)\cdot\frac{\partial P_a(j)}{\partial w_t^k}-P_a(j)\cdot\frac{\partial P_d(j)}{\partial w_t^k}\right) \qquad for\ k=0,1,...,L \tag{8}$$

$$w_s^{k+1}=w_s^k-\alpha\cdot\sum_{j=0}^{1}\left(P_d(j)\cdot\frac{\partial P_a(j)}{\partial w_s^k}-P_a(j)\cdot\frac{\partial P_d(j)}{\partial w_s^k}\right) \qquad for\ k=0,1,...,L \tag{9}$$

The condition $P_d > P_a$ in (7) requires that only the unresolved path difference segments j are used in the optimization. Gradient descent, therefore, makes changes to the weight vectors in the direction that reduces the error measure E . Parameter α in (8),(9) is a fixed step size that we selected empirically at 0.003. Note that partial derivatives of cumulative path costs P_d and P_a in (8),(9) translate simply into partial derivatives of cost functions C_t and C_s which can be readily calculated without resorting to numerical methods.

After L iterations the PD gradient descent procedure corrects some or possibly all path difference segments. The resulting weight vectors are then used in (1),(2) to compute the globally optimal path again. This generally results in new previously unknown path difference segments. PD learning proceeds by appending the new path difference segments to the list. This process repeats for several rounds until no new path differences are discovered (figure 4). Fortunately, experiments with real guitar fingering data show that a relatively small number of such rounds are necessary before path difference list stabilizes. Furthermore, the stable path difference list contains only a small fraction of all possible dynamic programming states therefore maintaining the advantage of low computational complexity. We are now able to summarize the complete PD learning procedure in a table of algorithmic steps:

Step-1	Initialize weight vectors as some w_t^0 and w_s^0. Initialize path difference list as an empty list.
Step-2	Calculate the optimal path using (1),(2). Extract path differences relative to the desired path (figure 3)
Step-3	If no path difference segments are found, terminate learning. The result is the latest w_t^k , w_s^k
Step-4	Add unique path difference segments to the path difference list
Step-5	Run gradient descent for L^k iterations (8),(9). Dependency on k allows us to gradually increase L in later rounds.
Step-6	Repeat from step-2.

Table 1. PD learning algorithm

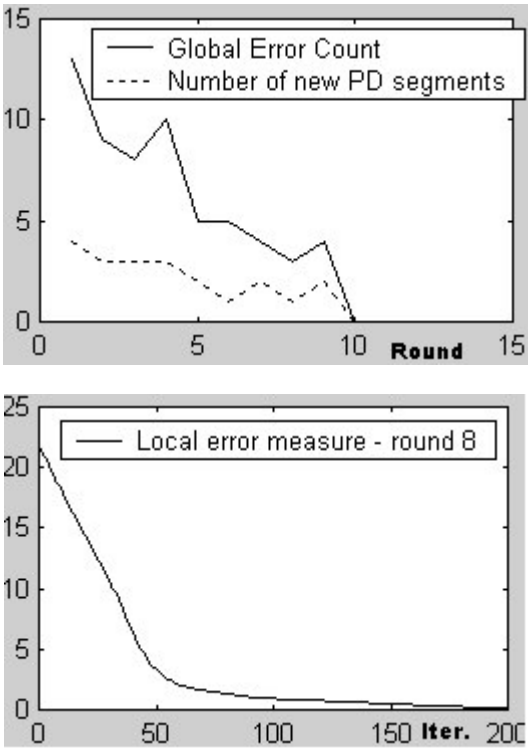

Figure 4. PD learning for composition *Adagio* by Albinoni. Learning runs for 10 rounds each with 200 gradient descent iterations. The bottom graph, a snapshot for round 8, shows the decrease in error E due to gradient descent.

5 Experiments and Results

We selected 7 classical guitar pieces and used published guitar tablatures as the reference for training and evaluation. The songs were then shortened to remove repeated sections since they do not contribute new training information. For initialization we used the best possible cost function weights we could obtain via an educated guess and manual tuning. Results are presented as error count, i.e. the number of stages where optimal path differs from the desired path. (see table 2 and 3),

Composition, Composer	Total stages	Initial Errors	Final Errors
Adagio, Albinoni	105	13	0
All My Trials, Greene and Carter	90	4	0
Desde el Alma, R. Melo	100	10	0
Love Story, Lai and Sigman	118	20	2
Moonlight Sonata, Beethoven	182	21	1
Allegretto, Mozart	75	2	0
Sch. Mexicano,M. Ponce	77	21	8

Table 2. Training results for 7 selected classical guitar compositions. PD learning was applied to each composition individually.

Composition	Total stages	Initial Errors	Final Errors
All 7 compositions	747	91	33

Table-3, Training results for 7 selected classical guitar compositions. PD learning was applied to all compositions combined.

Overall, PD method of learning cost function weights achieves the desired adaptation to a given guitar playing style. Training on individual songs (table 2) shows excellent adaptation but the results do not perform well on a test set due to insufficient training data. On the other hand, training with the combined data set (table 3) yields good generalization while significantly reducing the transcription error rate. Note, however, that a complete adaptation is not achieved due to several possible reasons: 1. desired paths are not labeled consistently by the expert guitarists, 2. the cost function features do not contain all the information involved in decision making and 3. the linear structure of the cost function does not provide sufficient flexibility. In summary, we conclude that PD learning meets the desired objective as it consistently outperforms the manual method of tuning the cost function weights.

As a side note, when multi-layer feed-forward neural network is used for functional approximation of cost functions C_t and C_s the partial derivatives in (8),(9) translate into the neural backpropagation algorithm (Hertz, Palmer, Krogh 1991). We expect the neural approximators to perform better than linear, however, care must be taken to avoid overfitting and lack of generalization.

References

Bertsekas, P. D. (1987). *Dynamic Programming*, Prentice Hall, New Jersey.

Bertsekas, P.D., J.N.Tsitsiklis, (1995). "Neuro-Dynamic Programming: An Overview", *Proceedings of the 34th Conference on Decision & Control*, New Orleans, LA.

Goldberg, D., (1989). Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Publishing Co.

Gilardino, A.,(1975) "Il Problema della ditteguitara nelle Musiche per Chitarra," *Il Fronimo* 10.

Harmony Central, (2004), www.harmony-central.com

Harris, J.,(1999). *50 Classical Guitar Pieces - In Tablature and Standard Notation*, Creative Concepts.

Hertz, A. J., R.G. Palmer, A.S. Krogh, (1991). *Introduction to the theory of neural computation*, Addison-Wesley Publishing Co.

Kirkpatrick S., Gelatt C. D. and Vecchi M. P., "Optimization by simulated annealing", Science, May 1983, pp. 220 (4598)

OLGA, (2004), On-Line Guitar Archive, www.olga.net

Phillips,M., John Chappel, Jon Chappel, (1998). *Guitar for Dummies*, Hungry Minds.

Sayegh,S.I., (1989). "Fingering for String Instruments with the Optimum Path Paradigm", *Computer Music Journal*, vol.13, No. 3, Fall 1989, pp. 76-83.